2021初赛讲解

信智学院　陈琰宏

1 单项选择题

1. 以下不属于面向对象程序设计语言的是（D ）。

A. C++　　B. Python　　C. Java　　D. C

2. 以下奖项与计算机领域最相关的是（B ）。

A. 奥斯卡奖　　B. 图灵奖　　C. 诺贝尔奖　　D. 普利策奖

3. 目前主流的计算机储存数据最终都是转换成（A ）数据进行储存。

A. 二进制　　B. 十进制　　C. 八进制　　D. 十六进制

4. 以比较作为基本运算，在 N 个数中找出最大数，最坏情况下所需要的最少的比较次数为 （C ）。

A. N^2　　B. N　　C. N-1　　D. N+1

初始化第一个数为最大值，每个数都比较过，为n-1次

5. 对于入栈顺序为 a, b, c, d, e 的序列，下列（ D）不是合法的出栈序列。

A. a, b, c, d, e　　B. e, d, c, b, a

C. b, a, c, d, e　　D. c, d, a, e, b

c b a的顺序不能变

1 单项选择题

6.对于有 n 个顶点、m 条边的无向连通图 (m>n)，需要删掉（D ）条边才能使其成为一棵树。

A.n-1　　B. m-n　　C. m-n-1　　D. m-n+1

n顶点的连通图最小需要 n-1 的点，所以去掉 m-n+1条边

7.二进制数 101.11 对应的十进制数是（ C）。

A. 6.5　　B. 5.5　　C. 5.75　　D. 5.25

8.如果一棵二叉树只有根结点，那么这棵二叉树高度为 1。请问高度为 5 的完全二叉树有 （A ）种不同的形态？

A. 16　　B. 15　　C. 17　　D. 32

因为是完全二叉树，只要算第5层的元素即可，第五层的元素个数为$2^{n-1}=2^4=16$

9.表达式 a*(b+c)*d 的后缀表达式为(B)，其中 * 和 + 是运算符。

A. **a+bcd　　B. abc+*d*　　C. abc+d**　　D.*a*+bcd

1 单项选择题

10. 6 个人，两个人组一队，总共组成三队，不区分队伍的编号。不同的组队情况有（ B ）种。

A. 10　　B. 15　　C. 30　　D. 20

C(6,2)*C(4,2)*C(2,2)/3!=(15*6)/3!=15

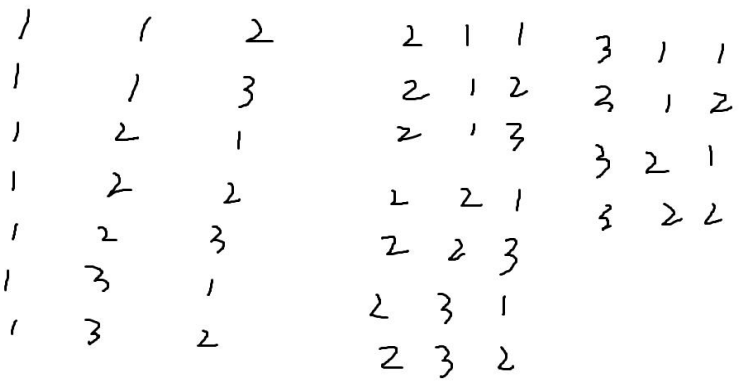
区分队伍编号的情况/重复情况

11. 在数据压缩编码中的哈夫曼编码方法，在本质上是一种（B ）的策略。

A. 枚举　　B. 贪心　　C. 递归　　D. 动态规划

12. 由 1,1,2,2,3 这五个数字组成不同的三位数有（A ）种。

A. 18　　B. 15　　C. 12　　D. 24



1 单项选择题

13.考虑如下递归算法

```
solve(n)
      if n<=1 return 1
       else if n>=5 return n*solve(n-2)
       else return n*solve(n-1)
```

则调用 solve(7) 得到的返回结果为（ C ）。

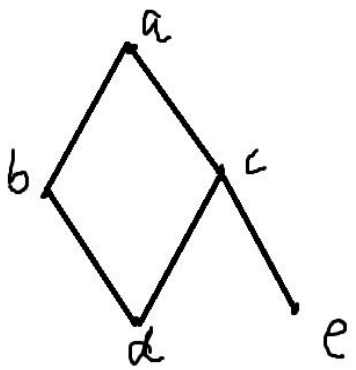
A. 105　　B. 840　　C. 210　　D. 420

7*5*3*2*1=210

1 单项选择题

14. 以 a 为起点，对右边的无向图进行深度优先遍历，则 b, c, d, e 四个点中有可能作为
最后一个遍历到的点的个数为（ ）。

A. 1　　B. 2　　C. 3　　D. 4



abcde， acedb，

15. 有四个人要从 A 点坐一条船过河到 B 点，船一开始在 A 点。该船一次最多可坐两个人。 已知这四个人中每个人独自坐船的过河时间分别为 1, 2, 4, 8，且两个人坐船的过河时 间为两人独自过河时间的较大者。则最短（ ）时间可以让四个人都过河到 B 点（包括从 B 点把船开回 A 点的时间）。

A. 14　　B. 15　　C. 16　　D. 17

贪心策略，凑。
4 8 过河，1 2驶船从B回到A. 故
1 2过河 1返回，花费时间3； 4 8过河，2返回，花费时间10；1,2过河，花费时间2.

2 阅读程序写结果1[6330]

```
1  #include <iostream>
2  using namespace std;
3  int n;
4  int a[1000];
5  int f(int x)
6  {//统计x 二进制补码表示下1的个数，对负数依然适用
7      int ret = 0;
8      for(;x;x&=x-1)ret++;
9      return ret;
10 }
11 int g(int x)
12 {//求出x 二进制表示下最低位的1的位置
13     return x & -x;
14 }
15
```

```
16 int main( )
17 {
18     cin>> n;
19     for(int i=0;i<n;i++)cin>>a[i];
20     for(int i=0;i<n;i++)
21     cout<< f(a[i]) + g(a[i]) << ' ';
22     cout << endl;
23     return 0;
24 }
```

输入n个数，求出将每个数转化为二进制后，该数中1的个数以及 该数最低位1的位置 之和。

如数字5, 的二进制为101,101含有2个1，最低位的1的位置为1（从左往右三个位置分别是1,2,3）。计f(101)=2, g(101)=1,则f(5)+g(5)=3

另如：

11的二进制数为1011

f(1101)=3　　g(1011)=1

位运算

求出x 在二进制表示下最低位的1的位置

负数的二进制运算为补码运算

$-x=\sim x+1$

$x\&-x=x\&(\sim x+1)$

假设 x=10101...100000

~x=01010...011111

+1

100000

```
11 int g(int x)
12 {//求出x 二进制表示下最低位的1的位置
13     return x & -x;
14 }
15
```

x&-x=

假设 x=10101...100000

~x=01010...100000

00000...100000

位运算

统计x在二进制补码表示下1的个数

```
5  int f(int x)
6  {//统计x 二进制补码表示下1的个数，对负数依然适用
7      int ret = 0;
8      for(;x;x&=x-1)ret++;
9      return ret;
10 }
```

负数的二进制运算为补码运算

假设 x=1010110

&x-1=1010101

1010100（取出最低位）

&x-1=1010011

1010000（从低到高取出每个1）

2 阅读程序写结果1

判断题

（1）输入的 n 等于 1001 时，程序不会发生下标越界。（ F）

a 数组定义长度为 1000.因此下标范围为 0~999

（2）输入的 a[i] 必须全为正整数，否则程序将陷入死循环。（F ）

f 函数的作用是统计传入参数在二进制补码表示下 1的个数,对负数依然适用

（3）当输入为 5 2 11 9 16 10 时，输出为 3 4 3 17 5。（ F）

g 函数的作用是提取出传入参数在二进制补码表示下最低位的 1,因此 f(10)+g(10)=2+2=4.

（4）当输入为 1 511998 时，输出为 18。（ T）

511998 的二进制补码表示为 11111001111111111110，因此 f(511998) = 16g(511998)=2。

2 阅读程序写结果1

（5）将源代码中 g 函数的定义（14~ 17 行）移到 main 函数的后面，程序可以正常编译运行。（ F）

主函数前 g 函数缺少函数声明,无法在主函数中使用,会报错。

单选题

（6）当输入为 2 -65536 2147483647 时，输出为（ B）。

A. 65532 33　　B. 65552 32　　C. 65535 34　　D. 65554 33

非负数的补码表示与其二进制表示相同,负数的补码表示为其绝对值的二进制表示取反并加1。因为65536=2^{16} 所以-65536的补码表示为11111111111111110000000000000000,由此可知(-65536)=16，g(-65536)=65536；因为2147483647=2^{31}-1所以214748347的补码表示为01111111111111111111111111111111,因此f(2147483647)=31,g(2147483647)=1

2 阅读程序写结果2[6331]字符解密

```
#include <iostream>
#include <string>
using namespace std;
char base[64];
char table[256];
void init()
{
    for(int i=0;i<26;i++)base[i]='A'+i;
    for(int i=0;i<26;i++)base[26+i]='a'+i;
    for(int i=0;i<10;i++)base[52+i]='0'+i;
    base[62] = '+', base[63]= '/';
    for (int i = 0; i < 256; i++) table[i] = 0xff;
    for (int i = 0; i < 64; i++) table[base[i]] = i;
    table['='] = 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;
char base[64];  //char，1个字节
char table[256];
void init(){
    for (int i = 0; i < 26; i++) base[i] = 'A' + i;  //大写字母asc码存在base[0]-base[25]
    for (int i = 0; i < 26; i++) base[26 + i] = 'a' + i; //小写字母asc码存base[26]-base[51]
    for (int i = 0; i < 10; i++) base[52 + i] = '0' + i;//字符0-9asc码存base[52]-base[61]
    base[62] = '+', base[63] = '/';
    for (int i = 0; i < 256; i++) table[i] = 0xff;  //初始化，初始值是0xff
    for (int i = 0; i < 64; i++) table[base[i]] = i;  //更新，table[x]=y，asc码为x的字符
    table['='] = 0;   //在base[]数组的对应下标y
}
```

		[0]	[1]	[2]	[3]	…	[23]	[24]	[25]
char类型	base[ ]	'A'	'B'	'C'	'D'	…	'X'	'Y'	'Z'
		[26]	[27]	[28]	[29]	…	[49]	[50]	[51]
char类型	base[ ]	'a'	'b'	'c'	'd'	…	'x'	'y'	'z'
		[52]	[53]	[54]	[55]	…	[59]	[60]	[61]
char类型	base[ ]	'0'	'1'	'2'	'3'	…	'7'	'8'	'9'
		[62]	[63]						
char类型	base[ ]	'+'	'/'						

table[x]=y，asc码为x的字符在base[ ]数组的对应下标y

		'A'	'B'	'C'	'D'	…	'X'	'Y'	'Z'
char类型	table[ ]	0	1	2	3		23	24	25
		'a'	'b'	'c'	'd'	…	'x'	'y'	'z'
char类型	table[ ]	26	27	28	29		49	50	51
		'0'	'1'	'2'	'3'	…	'7'	'8'	'9'
char类型	table[ ]	52	53	54	55		59	60	61
		'+'	'/'	'='					
char类型	table[ ]	62	63	0					

除去以上65个字符外，其他字符的table[ ]=0xff

base64 是网络上最常见的用于传输8 Bit 字节码的编码方式之一，他的解码规则是把4个字节变成3个字节。解码得到的是任意的，没有限定。

```
16 string decode(string str)
17 {
18     string ret;
19     int i;
20     for (i = 0; i < str.size(); i+= 4) {
21         ret += table[str[i]] << 2| table[str[i + 1]]>> 4;
22         if (str[i + 2] != '=')
23         ret += (table[str[i + 1]] & 0xff)<< 4 | table[str[i +2]] >> 2;
24         if (str[i + 3] != '=')
25         ret += table[str[i + 2]]<<6 | table[str[i + 3]];
26     }
27 return ret;
28 }
29 int main()
30 {
31     init();
32 //  cout << int(table[0]) << endl;
33     string str;
34     cin>>str;
35     cout<< decode(str) << endl;
36     return 0;
37 }
```

```
string decode(string str){
    string ret;
    int i;
    for (i = 0; i < str.size(); i += 4) {   //每4个字符操作一轮
        ret += table[str[i]] << 2 | table[str[i + 1]] >> 4;   //char转化string
        if (str[i + 2] != '=')
            ret += (table[str[i + 1]] & 0x0f) << 4 | table[str[i + 2]] >> 2;
        if (str[i + 3] != '=')
            ret += table[str[i + 2]] << 6 | table[str[i + 3]];
    }
    return ret;  //返回字符串
}
```

判断题

（1）输出的第二行一定是由小写字母、大写字母、数字和 +、 /、= 构成的字符串。（ F ）

因为bse[64]只对64个字符进行了编码,所以如果是除此之外的字符,输出不确定。 比如字符‘*’,不在编码的字符中,如果输入4个“**”输出是不可显示的乱码。

输入“****”			
str[0]='*'	table['*']=0xff	table[str[0]]<<2=11111100	
str[1]='*'	table['*']=0xff	table[str[1]]>>4=11111111	
备注：对于signed char的负数，有逻辑右移和算术右移，编译器相关			
逻辑右移，左侧空位用0补齐，算术右移，左侧空位用1补齐，保持负数			
table[str[0]] << 2 | table[str[1]] >> 4		11111100	
		按位或	11111111
			11111111
asc码常用字符的编码是0-127			
255对应的char不是可显示字符，转化成string类型，添加到ret后面，是乱码			

（2）可能存在输入不同，但输出的第二行相同的情形。（ ）

输入的字符不在编码的字符范围内,就有可能输人入不同, 输出相同。比如“***" 和 “%%%”,table[‘*’]=tale[‘%’]Oxff,所以,对应的ret相回， 输出相同。

（3）输出的第一行为 -1。（ ）

table[0]=0xff
table[ ]的类型是char，即signed char，1个字节， int，即signed int，4个字节
signed char强制类型转化成int，则在前面的三个字节补符号位
补上0xffffff（char的首位为1）
或0x000000（char的首位为0）
unsigned char转换为int，则前面补上0x000000
所以转化后的数值是0xffffffff
是-1的补码

单选题

（4）设输入字符串长度为 n，decode 函数的时间复杂度为（ B）

A. O(√n)　　B. O(n)　　C. O(nlogn)　　D. O(n^2)

分析decode函数的for循环次数n/4,复杂度是O(n)。

（5）当输入为 Y3Nx 时，输出的第二行为（）。

A. csp　　B. csq　　C. CSP　　D. Csp

		[0]	[1]	[2]	[3]
str="Y3Nx"	str[]	'Y'	'3'	'N'	'x'
table[str[0]]=24		24的2进制:	00011000	左移两位	01100000
table[str[1]]=55		55的2进制:	00110111	右移4位	00000011
				按位或	01100011
10进制99	asc码99对应的字符是c				
table[str[1]]=55		55的2进制:	00110111		
		按位与	00001111		
			00000111	左移4位	01110000
table[str[2]]=13		13的2进制:	00001101	右移2位	00000011
				按位或	01110011
10进制115	asc码115对应的字符是s				
table[str[2]]=13		13的2进制:	00001101	左移6位	01000000
table[str[3]]=49		49的2进制:	00110001		00110001
				按位或	01110001
10进制113	asc码113对应的字符是q				

需要背asc码，或者发现113是115前面2个的字符

（6）当输入为Y2NmIDIwMjE= 时，输出的第二行为（ ）。

A. ccf2021　　B. ccf2022　　C. ccf 2021　　D. ccf 2022

2 阅读程序写结果3[6332]求约数和

```
#include <iostream>
using namespace std;
const int n = 100000;
const int N=n+1;
int m;
int a[N], b[N], c[N],d[N];
int f[N], g[N];
void init()
{
    f[1]=g[1]=1;
    for(int i=2;i<=n;i++){
        if (!a[i]) {
            b[m++] = i;
            c[i]=1,f[i]=2;
            d[i] = 1, g[i] =i+ 1;
        }
```

```
        for(int j=0;j<m&&b[j]*i<=n;j++){
            int k = b[j];
            a[i *k] = 1;
            if(i%k==0) {
                c[i*k]=c[i] + 1;
                f[i*k]=f[i]/c[i*k]*(c[i*k]+1);
                d[i * k] = d[i];
                g[i*k]=g[i]*k+d[i];
                break;
            }
            else {
                c[i*k] =1;
                f[i*k]=2*f[i];
                d[i * k] = g[i];
                g[i *k]= g[i] * (k + 1);
            }
        }
    }
}
```

```
int main()
{
    init();
    int x;
    cin>>x;
    cout<< f[x] <<' ' << g[x] << endl;
    return 0;
}
```

```
#include <iostream>  //线性筛求质数，每个合数被它的最小质因子筛去
using namespace std;
const int n = 100000;
const int N = n + 1;
int m;  //质数个数，计数器
int a[N], b[N], c[N], d[N];  //a[i]数组值为1标识i是合数，初始为0全是质数。b[ ]存质数,b[0]开始
void init(){
    for (int i = 2; i <= n; i++) {  //从2开始
        if (!a[i])          b[m++] = i;   //若a[i]==0，i是质数，存入b[ ]，数量加一
       for (int j = 0; j < m && b[j] * i <= n; j++) { //从最小质数开始筛，无论i是质数、合数
            int k = b[j]; //k是已找到的比i小的质数，i是质数，乘以小于i的质数，新合数;i是合数，
            a[i * k] = 1; //i乘以比i小的质数，新合数，i*k是合数，标记，筛去
            if (i % k == 0) //若i不能被k整除，k不是i的质因子，k是i*k的最小质数，继续筛
                break; // i被k整除，k是i*k的最小质数且重复，退出。
       } //若继续，则合数不是被最小质因筛去（被更多的质因子也筛去），有重复
    }
}
```

每个合数被且仅被它的最小质因子筛去，没筛掉的就是质数
每个合数都只被筛一次，不会重复筛选，所以复杂度是O(n)。

	找到质数	b[0]=2	b[1]=3	b[2]=5	b[3]=7	b[4]=11
i=2	b[0]=2	2*2				
i=3	b[1]=3	3*2	3*3			
i=4		4*2				
i=5	b[2]=5	5*2	5*3	5*5		
i=6		6*2				
i=7	b[3]=7	7*2	7*3	7*5	7*7	
i=8		8*2				
i=9		9*2	9*3			
i=10		10*2				
i=11	b[4]=11	11*2	11*3	11*5	11*7	11*11

```
for (int i = 2; i <= n; i++) {
    if (!a[i])    b[m++] = i;
    for (int j = 0; j < m && b[j] * i <= n; j++) {
        int k = b[j];
        a[i * k] = 1;
        if (i % k == 0)
            break;
    }
}
```

i循环升序遍历i:

1.如果i是质数（a[i]==0），那么，质数i乘以小于i的质数，筛出来的合数与之前不会重复。

2.如果i是合数（a[i]==1），i可以表示成递增质数相乘：$i=p_1*p_2*\cdots*p_n$，p_i都是质数，p_1是最小的质数，筛出所有<=p1的质数*i，即i%p1==0时退出。

比如：i=2*3*5=30，筛出i*2=60，不能筛出i*3=90，90=2*3*3*5在i是45=3*3*5时筛出(45*2)，这样筛法不重复。

每个合数都可以分解成质因子的幂次的乘积

$X=p_1^{a1}*p_2^{a2}*\cdots*p_n^{an}$，pi都是质数，ai是幂次，$p_1$是最小的质数

约数个数：$(1+a_1)*(1+a_2)*\cdots*(1+a_n)$

$36=2*2*3*3=2^2*3^2$，约数个数$(1+2)*(1+2)=9$

2的幂次有3种情况	3的幂次有3种情况
2^0	3^0
2^1	3^1
2^2	3^2

乘法原理，36的约数有9个：1、2、3、4、6、9、12、18、36

```
#include <iostream>   //求约数个数
using namespace std;
const int n = 100000;
const int N = n + 1;
int m;
int a[N], b[N], c[N], d[N];  //c[i]:i的最小质因数的个数
int f[N], g[N];  //f[i]:i的约数的个数
void init(){
    f[1] = g[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!a[i]) {   //i是质数
            b[m++] = i;
            c[i] = 1, f[i] = 2;  //i的最小质因数是i,个数1个。i的约数2个：1和i
            d[i] = 1, g[i] = i + 1;
        }
```

```
    for (int j = 0; j < m && b[j] * i <= n; j++) {
        int k = b[j];
        a[i * k] = 1;
        if (i % k == 0) {  //k是i*k的最小质因子，i包括最小质因子k，c[i]是k的个数
            c[i * k] = c[i] + 1; // 在c[i]最小质因子k的个数基础上加一。
            f[i * k] = f[i] / c[i * k] * (c[i * k] + 1);
          // f[i*k]的最小质因子k比f[i]多1个
            XXXXXXXXX
        }
        else {  //i不能整除k
            c[i * k] = 1;  // i不包括最小质因子k，i*k的最小质因子k的个数1
            f[i * k] = 2 * f[i];  //
            //  f[i*k]的质因子比f[i]多一个k，且个数1，所以因数个数为f[i]*(1+1)
            XXXXXXX
        }
    }
  }
}
```

	$X=p_1^{a1}*p_2^{a2}*\cdots*p_n^{an}$	p_i是质因子，升序排序
X的约数和g[X]	$(p_1^0+p_1^1+\cdots+p_1^{a1})*(p_2^0+p_2^1+\cdots+p_2^{a2})*\cdots*(p_n^0+p_n^1+\cdots+p_n^{an})$	
d[X]是红色部分，g[X]连乘积中去除最小质因子连乘积的部分		
i是质数时：		
	d[i]=1	
	g[i]=1+i	i只有i这一个质因子
当i%k!=0	k不是i的质因子	
	d[i*k]=g[i]	i*k比i多一个质因子k，d[i*k]是g[i*k]中不包括k连乘积的部分
	g[i*k]=g[i]*(1+k)	约数和公式
当i%k==0	k是i的质因子，是i和i*k的最小质因子	
	d[i*k]=d[i]	根据定义，连乘积中不包括k连乘积的部分
	g[i*k]=g[i]*k+d[i]	
6=2*3, g[6]=(1+2)*(1+3)=12		d[6]=(1+3)=4
i=6，k=2时	i%k==0	
	$g[i*k]=g[12]=(1+2^1+2^2)*(1+3^1)$	
	$g[i]=g[6]=(1+2^1)*(1+3^1)$	
	$g[i]*k+d[i]=(1+2^1)*(1+3^1)*2+1*(1+3^1)$	

假设输入的 x 是不超过 1000 的自然数，完成下面的判断题和单选题：

判断题

(1)若输入不为1，把第 13 行删去不会影响输出的结果。（ ）

(2)(2 分) 第 25 行的 f[i] / c[i * k]可能存在无法整除而向下取整的情况。（ ）

(3)(2 分) 在执行完 init() 后，f 数组不是单调递增的，但 g 数组是单调递增的。（ ）

单选题

(4)init 函数的时间复杂度为（ ）。

A. O(n)　B. O(nlogn)　C. O(n√ n)　D. O(n^2)

(5)在执行完 init() 后,f[1],f[2],f[3]…f[100] 中有（）个等于 2。

A. 23　B. 24　C. 25　D. 26

(6)(4 分) 当输入为1000 时，输出为（）。

A. 15 1340　B. 15 2340　C. 16 2340　D. 16 1340

假设输入的 x 是不超过 1000 的自然数，完成下面的判断题和单选题：

判断题

(1)若输入不为1，把第 13 行删去不会影响输出的结果。（ ）

(2)(2 分) 第 25 行的 f[i] / c[i * k]可能存在无法整除而向下取整的情况。（ ）

(3)(2 分) 在执行完 init() 后，f 数组不是单调递增的，但 g 数组是单调递增的。（ ）

单选题

(4)init 函数的时间复杂度为（ ）。

A. O(n)　B. O(nlogn)　C. O(n√ n)　D. O(n^2)

(5)在执行完 init() 后,f[1],f[2],f[3]…f[100] 中有（）个等于 2。

A. 23　B. 24　C. 25　D. 26

(6)(4 分) 当输入为1000 时，输出为（）。

A. 15 1340　B. 15 2340　C. 16 2340　D. 16 1340

3 完善程序1 [6333] Josephus 问题

.(Josephus 问题） 有 n 个人围成一个圈，依次标号 0 至 n−1。从 0 号开始，依次 0,1,0,1,… 交替报数，报到 1 的人会离开，直至圈中只剩下一个人。求最后剩下人的编号。
试补全模拟程序。

```
#include <iostream>
using namespace std;
const int MAXN = 1000000 ;
int F[MAXN];
int main() {
    int n;
    cin>>n;
    int i=0,p=0,c=0;
    while (c < n-1) {
        if (F[i] == 0) {
        if ( p ) {
            F[i] = 1;
            c++;
        }
         p^=1;
        }
        i=(i+1)%n;
    }
    int ans = -1;
    for(i=0;i<n;i++)
    if (F[i] == 0)
        ans = i;
    cout << ans << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
const int MAXN = 1000000;
int F[MAXN];
int main() {
    int n;
    cin >> n;
    int i = 0, p = 0, c = 0;
    while (①) {    // c<n-1
        if (F[i] == 0) {
            if (②) { // p
                F[i] = 1;
                ③; //c++
            }
            ④; // p^=1
        }
        ⑤; // i=(i+1)%n
    }
    int ans = -1;
    for (i = 0; i < n; i++)  //答案是F[i]==0的i
        if (F[i] == 0)
            ans = i;
    cout << ans << endl;
    return 0;
}
```

//先填1，while内if(F[i]==0)，所以i不是计数器，c是计数器
//再填3，离开人数+1
//再填5，遍历F[]每一个下标，到了[n-1]下一个是[0]， while内if(F[i]==0)，i是下标
//再填2，p是控制0、1，初始值为0，p=1进if，找到人，再0\1变换，p在if(F[i]==0)内
//再填4 ，p^=1，值0、1变换

(1)①处应填（ ）

A.i < n　B.c < n　C.i < n- 1　D.c < n-1

(2)②处应填（ ）

A.i % 2 == 0　B.i % 2 == 1　C.p　D.!p

(3)③处应填（ ）

A.i++　B.i = (i + 1) % n　C.c++　D.p ^= 1

(4)④处应填（ ）

A.i++　B.i = (i + 1) % n　C.c++　D.p ^= 1

(5)⑤处应填（ ）

A.i++　B.i = (i + 1) % n　C.c++　D.p ^= 1

3 完善程序2 [6334]矩形计数

面上有 n 个关键点，求有多少个四条边都和 x 轴或者 y 轴平行的矩形，满足四个顶点都是关键点。给出的关键点可能有重复，但完全重合的矩形只计一次。

```
#include <iostream>
using namespace std;
struct point {
    int x, y, id;
};
bool equals(point a, point b) { //判断相等，与id无关
    return a.x == b.x && a.y == b.y;
}
bool cmp(point a, point b) { // 升序或降序，选项都是升序
    return ①; // 第2：暂时选项B/D都无问题。
}
```

```
void sort(point A[], int n) {  //冒泡排序
    for (int i = 0; i < n; i++)
        for (int j = 1; j < n; j++)
            if (cmp(A[j], A[j - 1])) {  //若A[j]更小，交换
                point t = A[j];
                A[j] = A[j - 1];
                A[j - 1] = t;
            }
        }
int unique(point A[], int n) {  //剔重
        int t = 0;
        for (int i = 0; i < n; i++)
            if (②)    //第1：如果不等，存入A[t]，A[i]与已存的A[t-1]比，考虑第1个数
                A[t++] = A[i];  //上：  t == 0 || !equals(A[i], A[t - 1])，只有这项不等于
      return t;
}
```

4 完善程序2

```
bool binary_search(point A[], int n, int x, int y) {  //二分答案
    point p;  //查找目标
    p.x = x;
    p.y = y;
    p.id = n;  //id最大
    int a = 0, b = n - 1;  //a、b分别是二分区间左右边界
    while (a < b) { // A[]升序
        int mid = ③;  // 第3：(a+b)>>1，右移运算符
        if (④)  //第3：二分。如果去右区间，说明A[mid]<p
            a = mid + 1;  //上：cmp(A[mid], p)
        else
            b = mid;
    }
    return equals(A[a], p);  //判断相等的equals函数没有id，所以38题是B，也不能有id
}  //否则，x/y相同的两个点是cmp小于关系
const int MAXN = 1000;
point A[MAXN];
```

4 完善程序2

```
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> A[i].x >> A[i].y;
        A[i].id = i;
    }
    sort(A, n);  //排序
    n = unique(A, n);  //剔重
    int ans = 0;
    for (int i = 0; i < n; i++)  //枚举点对
        for (int j = 0; j < n; j++)  //第4：剔重，  D. A[i].x < A[j].x && A[i].y < A[j].y
            if (⑤ && binary_search(A, n, A[i].x, A[j].y) &&   //见下页
                binary_search(A, n, A[j].x, A[i].y)) {
                    ans++;
            }
    cout << ans << endl;
    return 0;
}
```

4 完善程序2

4 完善程序2

今天的课程结束啦……

下课了…
同学们再见！